Aalto University

CS-C3240 - Machine Learning D

Predicting Password Metrics Using Machine Learning

Stage 2

1. Introduction

In the modern world, more than 5 billion people are connected to the internet daily [1]. As an ever-increasing part of our lives becomes intertwined with digital reality, the more potential targets there are on the internet for hackers. Cyber security attacks pose a great risk for both private users and companies as a whole. The cost of all cybercrimes for companies worldwide is estimated to be $10.5 trillion annually by 2025 [2]. The repercussions of cybersecurity attacks on private people in the most serious situations can include identity theft and financial loss. Many of the attacks succeed due to insufficient security measures, such as passwords that are too weak. This paper studies how different metrics, such as entropy, length, and invulnerability factors can be used to predict the strength of a password. The results could be beneficial for situations where the strength of a password needs to be evaluated.

This paper is organized in the following format. Section 1 introduces the topic and issue at hand. Section 2 discusses the problem in further detail and the available data used in this project. In Section 3 we will focus on the models chosen to represent the data, how the data is processed and how the data is divided. Section 4 then presents and compares the results of both of the methods. Finally, in Section 5, the conclusions are summarized, and potential improvements are discussed.

2. Problem Formulation

In this paper we are trying to evaluate **how to predict the cracking time of a password based on its entropy and strength**. The data points represent passwords, which are expressed as strings. The dataset contains 25 000 unique values for passwords.

Each datapoint is described with six attributes. The most relevant ones in this paper include the following attributes: strength, entropy, and crack_time_sec. The dataset also contains attributes called class_strength and length but later we will discuss the reason as to why they will not be a part of our paper. Entropy and strength will be the features in this paper. Both of the features have a numeric value, strength and entropy represented as floating-point numbers. The strength feature indicates the strength of a password as a floating-point number ranging from 0 to 1. The entropy feature describes the complexity and uniqueness of a password, with a range from 8.0 to 377.0.

As a label we have crack_time_sec. Crack_time_sec represents the estimated time it would take to crack the password, represented by a floating-point number. Crack_time explains the same thing as crack_time_sec, but in more human, readable terms. Crack_time is represented by following terms: “instant”, “4.00 minutes”, “3.00 hours”, “3.00 days”, and “eternity”, depending on what crack_time_sec value corresponds to the best term. Because this is a class value, it will not be used as a label.

The dataset is obtained from Kaggle [3]. The origin of the dataset stems from a “rockyou.txt” file, which is a known leaked dataset. The dataset contains passwords that were leaked during the security breach that involved a website called “RockYou”. The incident happened in 2009 and the leaked passwords have since been used for many research and analysis purposes.

Feature	Explanation	Data Type
strength	Represents the strength of a password	Floating-point number
entropy	Represents the uniqueness and complexity of a password	Floating-point number
crack_time_sec (label set)	Represents the time estimate for cracking a password	Floating-point number

Table 1. Dataset features, their explanations and types

3. Methods

3.1 Overview of Datapoints and Features

Dataset produces 1 000 00 datapoints with no missing occurrences. The main features included in the learning model are strength and entropy. We do not consider feature class_strength for two reasons. First, class_strength is composed according to a scale provided as a link in the sources, based on the entropy calculation [3]. Second, entropy will propagate the same, but more accurate results than class_strength, since entropy is the numerical interpretation of class_strength. Feature entropy is calculated according to the formula **Entropy H = - $\sum$ x ∈ passwords (P(x) · log2 P(x)) ≤ log2 (number of possible passwords),** that is used in modern information security to estimate the strength of a password [5]. Features length and strength are strongly connected with entropy and in tandem, they define the crack_time_sec, which is the reason we initially would like to choose them as features. However, there is a problem in this tandem. Features length and entropy have a collinearity of 0.996472. The value is so close to one that it could affect the models that we use, thus we've decided to get rid of this feature. The next step was to make a standardization of data. This is a common step for datasets, whose values vary significantly. We used the StandardScaler() of the sklearn.prepocessing library for this purpose [6]. Also, because such a big leap in labels' values makes it difficult to visualize data, we will map all the values with the log of 10. For this project, we are going to be using two different methods, both of which are supervised learning.

3.2 The Polynomial Regression Model

The regression model is the most suitable machine learning method for our project. The assertion is based on the dataset we use. The label set known as "crack_time_sec" is composed of concrete numerical values in seconds. The datatype of "crack-time_sec" is float64 and it stays on the range of 10^-7 to 10^104. Since our dataset propagates variables that could be considered mathematically, it is rational to try out the Polynomial regression model. At the very end, to prepare the predicted data we will use the Principal Component Analysis (PCA) method to reduce the dimension of our test feature collection (array) to one and introduce how well our graph fits the points of the test label. In other words, we decrease hypothesis space during visualization.

Polynomial regression propagates overfitting on the specific degree of polynomial function. We've found out that the smallest error is achieved when the degree is equal to or less than 5. At this point the error propagated was 10^⁻10 or less. To compute the error, we used Mean Squared Error (MSE) as a loss function. The MSE model is used for representing the difference between predicted values and actual values, and therefore can be used for assessing the quality of these predictions [7]. We chose to use this loss function, because sklearn provides this method and the method itself is commonly used in many machine learning applications, making it a good method for comparing different models [8].

3.3 Multi-layer Perceptron

Multi-layer Perceptron is an adequate model for regression prediction models where an output is predicted based on a given set of inputs [9]. This model is a neural network, which uses three different types of layers: the input layer, the hidden layer(s) and the output layer. The aim of this model is to learn to understand and predict relationships between features and it is mostly used for predicting relationships in non-linear data. Our aim is to evaluate whether this kind of approach gives a more accurate representation of the underlying relationships between the chosen features than the previously used polynomial regression model. [10] While we assume that the relationships in our data are mostly linear, if this is not the case, with MLP we should be able to get more accurate results, and therefore we have chosen to employ it as a comparative alternative for the polynomial regression model. As a loss function, we are again going to use Mean Squared Error for the same reasons as previously. In addition, as MLP is sensitive to outliers, the use of MSE can

provide some benefits in this regard, as MSE puts a larger weight on outliers with huge errors due to the squaring that happens in the function [11].

When it comes to MLP, a common practice is to evaluate which number of hidden layers and neurons gives the most accurate predictions, as there is no universal solution for this problem [9]. Because we have only a few features, usually only one or two layers work the best [12]. According to our evaluation, the most appropriate number of hidden layers for our data set is either one or three, because we found out they propagate the least error. The number of neurons is crucial, since too few neurons can cause underfitting and too many overfitting [13]. It is advised that the number of neurons should be somewhere between the sizes of the input layer and the output layer [14], meaning that the best results for us should occur when the number of neurons is one or two. According to our evaluation, the most accurate results are obtained when the number of neurons is three for this specific data set.

3.4 Splitting the Dataset

In the project we will use the tools provided by sklearn. More concretely sklearn.test_train_split – function with the distribution 2/3 for training and 1/3 for testing. We will also include shuffle attribute in the function to mix the data of dataset and test training on the different values. The main purpose of splitting the dataset is to assess the performance of the chosen model for this project. The splitting of the dataset into two parts, 2/3 for training and 1/3 for testing, serves an important function. First of all, we want to avoid overfitting in the sense that we do not want to test the model by using the same data as for training. So, by making 2/3 of the data available to be for training, we still have a lot of unseen data available for testing. This partition also makes it easier for us to see how well the model performs on the testing data. The training data is significantly larger than the testing data. This is for the reason that the model would learn to predict patterns more accurately. Also, the ratio of the data split that we have chosen is quite a standard one, and therefore seems like a good fit for our model. [15]

We did not split the dataset into three parts, with distinct training, validation and test sets, because the dataset contains many similar values and the validation using similar values would be pointless. We tried to find other suitable datasets online for validation, but unfortunately all datasets available were also based on the same “RockYou” website’s data breach and were therefore not suitable to be used for validation. Therefore, we only have training and testing sets.

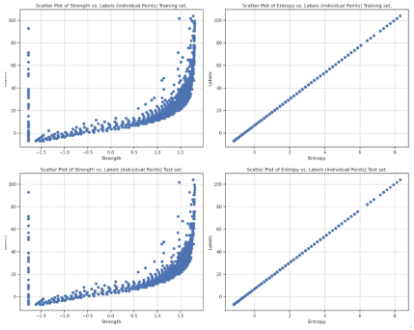
4. Results

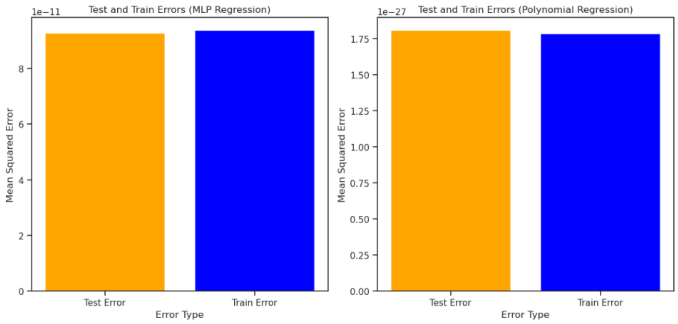
The results of our study were undoubtedly unusual. Both polynomial and MLP regressions propagated the model which has an accuracy almost 100%. Our further studies of the dataset, however, explained the result.

The dataset that we’ve been using to train and test our models appears to be linear. From figure 1 we can observe the dependency of a single feature to one’s output value. At the case of entropy, it is simply linear. Strength, however, diversify the dataset a little, but still has a big collinearity with entropy, which affects the outcome. The second reason is a noticeable repetition of the same input and output values which can be also seen from figure 1, or from the dataset itself. This additional reason makes our dataset very simple.

Because of these two main reasons, and especially because of the simplicity of the dataset, the Polynomial Regression model becomes almost linear. Thus, both error outcomes approach the value of zero. Standardization of data additionally decreases both training and testing errors by about 10^8 from $10^{-}16$ and thus, usual outcome for both the training and testing errors is 10^-(24-28). MLP Regression provides sufficiently worse result compared to the Polynomial Regression. However, since the training/test error result on average varies from 10^-9 to 10^{-}(9-12), it’s still excellent in the sense of the error’s value. MLP

model was clearly affected by the dataset, because it is supposed to be fitted by non-linear data in general. [16] The second reason is the scaling of the features. According to the documentation of sklearn, which we used to make a MLP regression, it propagates noises to outcome and thus increases errors [17].





Figures 1 and 2 respectively

Based on the error values for both validation and training sets, accuracy of the created model on the specific degree, dataset's simplicity and quality we claim that the Polynomial regression is more appropriate for our project. There is no overfitting nor underfitting in the training and test sets, which can be seen from Figure 2. To clarify, this is because of the simplicity of dataset and high enough amount of datapoints.

5. Conclusion

For both methods, the accuracy proved to be above 99%, which means that both methods used provide very accurate results for the strengths of the given passwords. Overall, the Polynomial Regression model seems to be more suitable for this type of data, since the observed testing error was significantly smaller than the error obtained from MLP. This result stems from the fact that the relationships in the data are linear, and the data set itself is very simple with many similar values, and therefore better suited for Polynomial Regression.

The problem we've studied has a few issues. First, lack of more features including capacity of cracking device, type of cracking attack, interpretation of how password is stored and how it is encrypted. The second is high collinearity between the features that we obtain, and high similarity index among the datapoints.

In addition to these problems, there was some room for limitations in the methods themselves. According to sklearn documentation, MLP can be affected by the standardization of data, which is crucial in our case. Another limitation was about the non-linearity of dataset and the feature number, since we have a linear dataset and only a few features. [17] Polynomial regression and MLP are both performing worse when features have a close to one collinearity value with respect to each other. Our project indeed suffered because of this problem. Choosing the best degree for Polynomial regression and the right number of layers is the big problem too, since we are basically just guessing them. There is obviously room for improvement in the usage of those methods. In our case it is surely adding new features and modifying the dataset. A second improvement could be to add more variety to the dataset. Third is decreasing collinearity between features.

References:

[1] Statista. Number of internet and social media users worldwide as of July 2023. Accessed 20.9.2023. https://www.statista.com/statistics/617136/digital-population-worldwide/

[2] Mike McLean. 9.8.2023. 2023 Must-Know Cyber Attack Statistics and Trends. Embroker. Accessed 20.9.2023. https://www.embroker.com/blog/cyber-attack-statistics/

[3] Karthik Udyawar. Password metrics dataset. Kaggle. Accessed 20.9.2023. https://www.kaggle.com/datasets/karthikudyawar/password-metrics-dataset

[4] Irma Šlekytė. 18.8.2023. Password Entropy: Definition and formula. Accessed 20.9.2023. https://nordvpn.com/blog/what-is-password-entropy/

[5] Subham Datta. 4.11.2022. How to Determine the Entropy of a Password. Accessed 20.9.2023. https://www.baeldung.com/cs/password-entropy

[6] sklearn.preprocessing.StandardScaler. scikit learn. Accessed 10.10.2023. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[7] Jim Frost. Mean Squared Error (MSE). Accessed 22.9.2023. https://statisticsbyjim.com/regression/mean-squared-error-mse/

[8] George Seif. 21.5.2019. Understanding the 3 most common loss functions for Machine Learning Regression. Accessed 22.9.2023. https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e14d3

[9] Jason Brownlee. 15.8.2022. When to Use MLP, CNN, and RNN Neural Networks. Machine Learning Mastery. Accessed 10.10.2023. https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/

[10] M.W Gardner and S.R Dorling. 8.1998. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. Atmospheric Environment. Accessed 10.10.2023. https://www.sciencedirect.com/science/article/pii/S1352231097004470

[11] George Seif. 21.5.2019. Understanding the 3 most common loss functions for Machine Learning Regression. Medium. Accessed 10.10.2023. https://towardsdatascience.com/understanding-the-3-most-common-loss-functions-for-machine-learning-regression-23e0ef3e14d3

[12] Harpreet Singh Sachdev. 23.1.2020. Choosing number of Hidden Layers and number of hidden neurons in Neural Networks. LinkedIn. Accessed 10.10.2023. https://www.linkedin.com/pulse/choosing-number-hidden-layers-neurons-neural-networks-sachdev

[13] Jason Brownlee. 6.8.2019. How to Configure the Number of Layers and Nodes in a Neural Network. Machine Learning Mastery. Accessed 10.10.2023. https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/

[14] Sandhya Krishnan. 8.9.2021. How do determine the number of layers and neurons in the hidden layer? Medium. Accessed 10.10.2023. https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3

[15] Data Science Wizards. 16.11.2022 A Guide to Data Splitting in Machine Learning. Accessed 22.9.2023. https://medium.com/@datasciencewizards/a-guide-to-data-splitting-in-machine-learning-49a959c95fa1

[16] sklearn.neural_network.MLPRegressor. scikit learn. Accessed 10.10.2023. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html

[17] 1.17. Neural network models (supervised). scikit learn. Accessed 10.10.2023. https://scikit-learn.org/stable/modules/neural_networks_supervised.html